PATENT COOPERATION TREATY

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF ELECTION

(PCT Rule 61.2)

To:

Commissioner
US Department of Commerce
United States Patent and Trademark Office, PCT
2011 South Clark Place Room CP2/5C24
Arlington, VA 22202
ETATS-UNIS D'AMERIQUE

in its capacity as elected Office

Date of mailing (day/month/year) 31 July 2001 (31.07.01)	
International application No. PCT/US00/23996	**Applicant's or agent's file reference** 10559-305WO1
International filing date (day/month/year) 31 August 2000 (31.08.00)	**Priority date** (day/month/year) 01 September 1999 (01.09.99)
Applicant WOLRICH, Gilbert et al	

1. The designated Office is hereby notified of its election made:

   [X] in the demand filed with the International Preliminary Examining Authority on:

   08 March 2001 (08.03.01)

   [ ] in a notice effecting later election filed with the International Bureau on:

2. The election [X] was [ ] was not

   made before the expiration of 19 months from the priority date or, where Rule 32 applies, within the time limit under Rule 32.2(b).

The International Bureau of WIPO **34, chemin des Colombettes** **1211 Geneva 20, Switzerland** Facsimile No.: (41-22) 740.14.35	Authorized officer Farid Abbou Telephone No.: (41-22) 338.83.38

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16716 A1

(51) International Patent Classification⁷: G06F 9/32, 9/44

(21) International Application Number: PCT/US00/23996

(22) International Filing Date: 31 August 2000 (31.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/151,961 1 September 1999 (01.09.1999) US

01 Mar 02 | 30 mos

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/151,961 (CIP)
Filed on 1 September 1999 (01.09.1999)

(71) Applicant *(for all designated States except US)*: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: WOLRICH, Gilbert [US/US]; 4 Cider Mill Road, Framingham, MA 01701 (US). ADILETTA, Matthew, J. [US/US]; 20 Monticello Drive, Worcester, MA 01603 (US). WHEELER, William [US/US]; 745 School Street, Webster, MA 01570 (US). BERNSTEIN, Debra [US/US]; 38 Helen Street, Waltham, MA 02452 (US). HOOPER, Donald [US/US]; 19 Main Circle, Shrewsbury, MA 01545 (US).

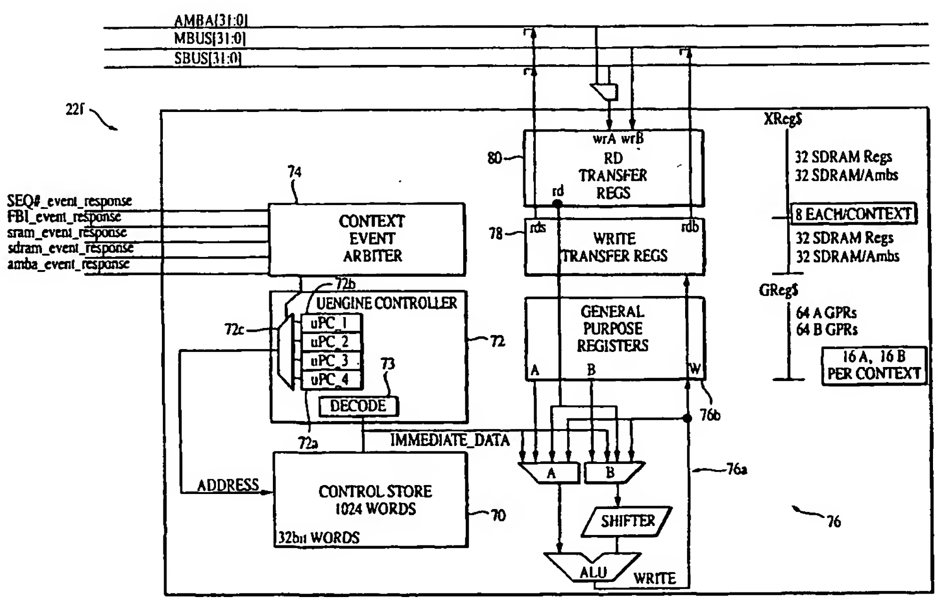
(74) Agents: MALONEY, Denis, G.; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 et al. (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE



(57) Abstract: A processor such as a parallel hardware-based multithreaded processor (12) is described. The processor (12) can execute a computer instruction that is a branch instruction that causes the instruction sequence in the processor (12) to determine whether a byte in a register (80, 78, 76b) is equal or not equal to a specified byte value contained in the instruction and performing the branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register (80, 78, 76b).

WO 01/16716 A1

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

BACKGROUND

This invention relates to branch instructions.

Parallel processing is an efficient form of information processing of concurrent events in a computing process. Parallel processing demands concurrent execution of many programs in a computer. Sequential processing or serial processing has all tasks performed sequentially at a single station whereas, pipelined processing has tasks performed at specialized stations. Computer code whether executed in parallel processing, pipelined or sequential processing machines involves branches in which an instruction stream may execute in a sequence and branch from the sequence to a different sequence of instructions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a communication system employing a processor.

FIG. 2 is a detailed block diagram of the processor.

FIG. 3 is a block diagram of a microengine used in the processor of FIGS. 1 and 2.

FIG. 4 is a diagram of a pipeline in the microengine.

FIG. 5 shows exemplary formats for branch instructions.

FIG. 6 is a block diagram of general purpose registers.

DESCRIPTION

Referring to FIG. 1, a communication system 10 includes a processor 12. In one embodiment, the processor is a hardware-based multithreaded processor 12. The processor 12 is coupled to a bus such as a PCI bus 14, a memory system 16 and a second bus 18. The system 10 is especially useful for tasks that can be broken into parallel sub-tasks or functions. Specifically hardware-based multithreaded processor 12 is useful for tasks that are bandwidth oriented rather than latency oriented. The hardware-based multithreaded processor 12 has multiple microengines 22 each with multiple hardware controlled threads that can be simultaneously active and independently work on a task.

The hardware-based multithreaded processor 12 also includes a central controller 20 that assists in loading microcode control for other resources of the hardware-based multithreaded processor 12 and performs other general purpose computer type functions such as handling protocols, exceptions, extra support for packet processing where the microengines pass the packets off for more detailed processing such as in boundary conditions. In one embodiment, the processor 20 is a Strong Arm® (Arm is a trademark of ARM Limited, United Kingdom) based architecture. The general purpose microprocessor 20 has an operating system. Through the operating system the processor 20 can call functions to operate on microengines 22a-22f. The processor 20 can use any supported operating system preferably a real time operating system. For the core processor implemented as a Strong Arm architecture, operating systems such as, MicrosoftNT® real-time, VXWorks and □CUS, a freeware operating system available over the Internet, can be used.

The hardware-based multithreaded processor 12 also includes a plurality of function microengines 22a-22f. Functional microengines (microengines) 22a-22f each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at any one time.

Microengines 22a-22f each have capabilities for processing four hardware threads. The microengines 22a-22f operate with shared resources including memory system 16 and bus interfaces 24 and 28. The memory system 16 includes a Synchronous Dynamic Random Access Memory (SDRAM) controller 26a and a Static Random Access Memory (SRAM) controller 26b. SDRAM memory 16a and SDRAM controller 26a are typically used for processing large volumes of data, e.g., processing of network payloads from network packets. The SRAM controller 26b and SRAM memory 16b are used in, e.g., networking packet processing, postscript processor, or as a processor for a storage subsystem, i.e., RAID disk storage, or for low latency, fast access tasks, e.g., accessing look-up tables, memory for the core processor 20, and so forth.

The processor 12 includes a bus interface 28 that couples the processor to the second bus 18. Bus interface 28 in one embodiment couples the processor 12 to the so-called FBUS 18 (FIFO bus). The processor 12 includes a second interface e.g., a PCI bus interface 24 that couples other system components that reside on the PCI 14 bus to

the processor 12. The PCI bus interface 24, provides a high speed data path 24a to the SDRAM memory 16a. Through that path data can be moved quickly from the SDRAM 16a through the PCI bus 14, via direct memory access (DMA) transfers.

Each of the functional units are coupled to one or more internal buses. The internal buses are dual, 32 bit buses (i.e., one bus for read and one for write). The hardware-based multithreaded processor 12 also is constructed such that the sum of the bandwidths of the internal buses in the processor 12 exceed the bandwidth of external buses coupled to the processor 12. The processor 12 includes an internal core processor bus 32, e.g., an ASB bus (Advanced System Bus) that couples the processor core 20 to the memory controller 26a, 26c and to an ASB translator 30 described below. The ASB bus is a subset of the so called AMBA bus that is used with the Strong Arm processor core. The processor 12 also includes a private bus 34 that couples the microengine units to SRAM controller 26b, ASB translator 30 and FBUS interface 28. A memory bus 38 couples the memory controller 26a, 26b to the bus interfaces 24 and 28 and memory system 16 including flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, each of the microengines 22a-22f includes an arbiter that examines flags to determine the available threads to be operated upon. Any thread from any of the microengines 22a-22f can access the SDRAM controller 26a, SDRAM controller 26b or FBUS interface 28. The memory controllers 26a and 26b each include a plurality of queues to store outstanding memory reference requests. The FBUS interface 28 supports Transmit and Receive flags for each port that a MAC device supports, along with an Interrupt flag indicating when service is warranted. The FBUS interface 28 also includes a controller 28a that performs header processing of incoming packets from the FBUS 18. The controller 28a extracts the packet headers and performs a microprogrammable source/destination/protocol hashed lookup (used for address smoothing) in SRAM.

The core processor 20 accesses the shared resources. The core processor 20 has a direct communication to the SDRAM controller 26a to the bus interface 24 and to SRAM controller 26b via bus 32. However, to access the microengines 22a-22f and transfer registers located at any of the microengines 22a-22f, the core processor 20 access the microengines 22a-22f via the ASB Translator 30 over bus 34. The ASB translator 30 can physically reside in the FBUS interface 28, but logically is distinct. The ASB Translator 30 performs an address translation between FBUS microengine transfer

register locations and core processor addresses (i.e., ASB bus) so that the core processor 20 can access registers belonging to the microengines 22a-22c.

Although microengines 22 can use the register set to exchange data as described below, a scratchpad memory 27 is also provided to permit microengines to write data out to the memory for other microengines to read. The scratchpad 27 is coupled to bus 34.

The processor core 20 includes a RISC core 50 implemented in a five stage pipeline performing a single cycle shift of one operand or two operands in a single cycle, provides multiplication support and 32 bit barrel shift support. This RISC core 50 is a standard Strong Arm® architecture but it is implemented with a five stage pipeline for performance reasons. The processor core 20 also includes a 16 kilobyte instruction cache 52, an 8 kilobyte data cache 54 and a prefetch stream buffer 56. The core processor 20 performs arithmetic operations in parallel with memory writes and instruction fetches. The core processor 20 interfaces with other functional units via the ARM defined ASB bus. The ASB bus is a 32-bit bi-directional bus 32.

Referring to FIG. 3, an exemplary microengine 22f includes a control store 70 that includes a RAM which stores a microprogram. The microprogram is loadable by the core processor 20. The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-72d. The four micro program counters 72a-72d are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ_#_EVENT_RESPONSE; FBI_EVENT_RESPONSE; SRAM _EVENT_RESPONSE; SDRAM _EVENT_RESPONSE; and ASB _EVENT_RESPONSE) from each one of the shared resources, e.g., SRAM 26a, SDRAM 26b, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a thread has completed and signaled completion, the thread needs to wait for that completion signal, and if the thread is enabled to operate, then the thread is placed on an available thread list (not shown). The microengine 22f can have a maximum of e.g., 4 threads available.

In addition to event signals that are local to an executing thread, the microengines 22 employ signaling states that are global. With signaling states, an executing thread can broadcast a signal state to all microengines 22. Receive Request or

Available signal, any and all threads in the microengines can branch on these signaling states. These signaling states can be used to determine availability of a resource or whether a resource is due for servicing.

The context event logic 74 has arbitration for the four (4) threads. In one embodiment, the arbitration is a round robin mechanism. Other techniques could be used including priority queuing or weighted fair queuing. The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions. The arithmetic logic unit includes condition code bits that are used by instructions described below. The registers set 76b has a relatively large number of general purpose registers that are windowed as will be described so that they are relatively and absolutely addressable. The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80. These registers are also windowed so that they are relatively and absolutely addressable. Write transfer register stack 78 is where write data to a resource is located. Similarly, read register stack 80 is for return data from a shared resource. Subsequent to or concurrent with data arrival, an event signal from the respective shared resource e.g., the SRAM controller 26a, SDRAM controller 26b or core processor 20 will be provided to context event arbiter 74 which will then alert the thread that the data is available or has been sent. Both transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path.

Referring to FIG. 4, the microengine datapath maintains a 5-stage micro-pipeline 82. This pipeline includes lookup of microinstruction words 82a, formation of the register file addresses 82b, read of operands from register file 82c, ALU, shift or compare operations 82d, and write-back of results to registers 82e. By providing a write-back data bypass into the ALU/shifter units, and by assuming the registers are implemented as a register file (rather than a RAM), the microengine can perform a simultaneous register file read and write, which completely hides the write operation.

The instruction set supported in the microengines 22a-22f support conditional branches. The worst case conditional branch latency (not including jumps) occurs when the branch decision is a result of condition codes being set by the previous microcontrol instruction. The latency is shown below in Table 1:

TABLE 1

```
                  | 1| 2| 3| 4| 5| 6| 7| 8|
-------------------+----+----+----+----+----+----+----+----+
microstore lookup | n1 | cb | n2 | XX | b1 | b2 | b3 | b4 |
reg addr gen      |    | n1 | cb | XX | XX | b1 | b2 | b3 |
reg file lookup   |    |    | n1 | cb | XX | XX | b1 | b2 |
ALU/shifter/cc    |    |    |    | n1 | cb | XX | XX | b1 |
write back        |    |    | m2 |    | n1 | cb | XX | XX |
```

where nx is pre-branch microword (n1 sets cc's), cb is conditional branch, bx is post-branch microword and XX is an aborted microword

As shown in Table 1, it is not until cycle 4 that the condition codes of n1 are set, and the branch decision can be made (which in this case causes the branch path to be looked up in cycle 5). The microengine 22f incurs a 2-cycle branch latency penalty because it must abort operations n2 and n3 (the 2 microwords directly after the branch) in the pipe, before the branch path begins to fill the pipe with operation b1. If the branch is not taken, no microwords are aborted and execution continues normally. The microengines have several mechanisms to reduce or eliminate the effective branch latency.

The microengines support selectable deferred branches. Selectable deferring branches are when a microengine allows 1 or 2 micro instructions after the branch to execute before the branch takes effect (i.e. the effect of the branch is "deferred" in time). Thus, if useful work can be found to fill the wasted cycles after the branch microword, then the branch latency can be hidden. A 1-cycle deferred branch is shown below in Table 2 where n2 is allowed to execute after cb, but before b1:

TABLE 2

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n2	XX	b1	b2	b3	b4
reg addr gen		n1	cb	n2	XX	b1	b2	b3
reg file lookup			n1	cb	n2	XX	b1	b2
ALU/shifter/cc				n1	cb	n2	XX	b1
write back					n1	cb	n2	XX

A 2-cycle deferred branch is shown in TABLE 3 where n2 and n3 are both allowed to complete before the branch to b1 occurs. Note that a 2-cycle branch deferment is only allowed when the condition codes are set on the microword preceding the branch.

TABLE 3

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	cb	n2	n3	b1	b2	b3	b4	b5
reg addr gen		n1	cb	n2	n3	b1	b2	b3	b4
reg file lkup			n1	cb	n2	n3	b1	b2	b3
ALU/shftr/cc				n1	cb	n2	n3	b1	b2
write back					n1	cb	n2	n3	b1

The microengines also support condition code evaluation. If the condition codes upon which a branch decision are made are set 2 or more microwords before the branch, then 1 cycle of branch latency can be eliminated because the branch decision can be made 1 cycle earlier as in Table 4.

TABLE 4

	1	2	3	4	5	6	7	8
microstore lookup	n1	n2	cb	XX	b1	b2	b3	b4
reg addr gen		n1	n2	cb	XX	b1	b2	b3
reg file lookup			n1	n2	cb	XX	b1	b2
ALU/shifter/cc				n1	n2	cb	XX	b1
write back					n1	n2	cb	XX

In this example, n1 sets the condition codes and n2 does not set the conditions codes. Therefore, the branch decision can be made at cycle 4 (rather than 5), to eliminate 1 cycle of branch latency. In the example in Table 5 the 1-cycle branch deferment and early setting of condition codes are combined to completely hide the branch latency. That is, the condition codes (cc's) are set 2 cycles before a 1-cycle deferred branch.

TABLE 5

	1	2	3	4	5	6	7	8
microstore lookup	n1	n2	cb	n3	b1	b2	b3	b4
reg addr gen		n1	n2	cb	n3	b1	b2	b3
reg file lookup			n1	n2	cb	n3	b1	b2
ALU/shifter/cc				n1	n2	cb	n3	b1
write back					n1	n2	cb	n3

In the case where the condition codes cannot be set early (i.e. they are set in the microword preceding the branch), the microengine supports branch guessing which attempts to reduce the 1 cycle of exposed branch latency that remains. By "guessing" the branch path or the sequential path, the microsequencer pre-fetches the guessed path 1 cycle before it definitely knows what path to execute. If it guessed correctly, 1 cycle of branch latency is eliminated as shown in Table 6.

TABLE 6

guess branch taken /branch is taken

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n1	b1	b2	b3	b4	b5
reg addr gen		n1	cb	XX	b1	b2	b3	b4
reg file lookup			n1	cb	XX	b1	b2	b3
ALU/shifter/cc				n1	cb	XX	b1	b2
write back					n1	cb	XX	b1

If the microcode guessed a branch taken incorrectly, the microengine still only wastes 1 cycle as in TABLE 7

TABLE 7

guess branch taken /branch is NOT taken

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n1	XX	n2	n3	n4	n5
reg addr gen		n1	cb	n1	XX	n2	n3	n4
reg file lookup			n1	cb	n1	XX	n2	n3
ALU/shifter/cc				n1	cb	n1	XX	n2
write back					n1	cb	n1	XX

However, the latency penalty is distributed differently when microcode guesses a branch is not taken. For guess branch NOT taken / branch is NOT taken there are no wasted cycles as in Table 8.

Table 8

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n1	n2	n3	n4	n5	n6
reg addr gen		n1	cb	n1	n2	n3	n4	n5
reg file lookup			n1	cb	n1	n2	n1	b4
ALU/shifter/cc				n1	cb	n1	n2	n3
write back					n1	cb	n1	n2

However for guess branch NOT taken /branch is taken there are 2 wasted cycles as in Table 9.

Table 9

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n1	XX	b1	b2	b3	b4
reg addr gen		n1	cb	XX	XX	b1	b2	b3
reg file lookup			n1	cb	XX	XX	b1	b2
ALU/shifter/cc				n1	cb	XX	XX	b1
write back					n1	cb	XX	XX

The microengine can combine branch guessing with 1-cycle branch deferment to improve the result further. For guess branch taken with 1-cycle deferred branch/branch is taken is in Table 10.

Table 10

	1	2	3	4	5	6	7	8
microstore lookup	n1	cb	n2	b1	b2	b3	b4	b5
reg addr gen		n1	cb	n2	b1	b2	b3	b4
reg file lookup			n1	cb	n2	b1	b2	b3
ALU/shifter/cc				n1	cb	n2	b1	b2
write back					n1	cb	n2	b1

In the case above, the 2 cycles of branch latency are hidden by the execution of n2, and by correctly guessing the branch direction.

If microcode guesses incorrectly, 1 cycle of branch latency remains exposed as in Table 11 (guess branch taken with 1-cycle deferred branch/branch is NOT taken).

Table 11

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	cb	n2	XX	n3	n4	n5	n6	n7
reg addr gen		n1	cb	n2	XX	n3	n4	n5	n6
reg file lkup			n1	cb	n2	XX	n3	n4	n5
ALU/shftr/cc				n1	cb	n2	XX	n3	n4
write back					n1	cb	n2	XX	n3

If microcode correctly guesses a branch NOT taken, then the pipeline flows sequentially in the normal unperturbed case. If microcode incorrectly guesses branch NOT taken, the microengine again exposes 1 cycle of unproductive execution as shown in Table 12.

Table 12

guess branch NOT taken/branch is taken

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	cb	n2	XX	b1	b2	b3	b4	b5
reg addr gen		n1	cb	n2	XX	b1	b2	b3	b4
reg file lkup			n1	cb	n2	XX	b1	b2	b3
ALU/shftr/cc				n1	cb	n2	XX	b1	b2
write back					n1	cb	n2	XX	b1

where nx is pre-branch microword (n1 sets cc's)
cb is conditional branch
bx is post-branch microword
XX is aborted microword

In the case of a jump instruction, 3 extra cycles of latency are incurred because the branch address is not known until the end of the cycle in which the jump is in the ALU stage (Table 13).

Table 13

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	jp	XX	XX	XX	j1	j2	j3	j4
reg addr gen		n1	jp	XX	XX	XX	j1	j2	j3
reg file lkup			n1	jp	XX	XX	XX	j1	j2
ALU/shftr/cc				n1	jp	XX	XX	XX	j1
write back					n1	jp	XX	XX	XX

Referring to FIG. 5, the microengines support various branch instructions such as those that branch on condition codes. In addition, the microengines also support a branch instruction that branches on a byte being equal or not equal to a specified byte. The branch on byte instruction "BR=BYTE" includes a byte_spec field.

BR=BYTE, BR!=BYTE

This branch instruction branches to an instruction at the specified label if a specified byte in a longword matches or mismatches the byte_compare_value. The br=byte instruction prefetches the instruction for the "branch taken" condition rather than the next sequential instruction. The br!=byte instruction prefetches the next sequential instruction. These instructions set the condition codes in the microengine.

Format: br=byte[reg, byte_spec, byte_compare_value, label#], optional_token
br!=byte[reg, byte_spec, byte_compare_value, label#], optional_token

Reg A is a context-relative transfer register or general-purpose register that holds the operand. Byte_spec Number specifies a byte in register to be compared with byte_compare_value. Valid byte_spec values are 0 through 3. A value of 0 refers to the rightmost byte.

A byte_compare_value is a value used for comparison. Valid byte_compare_values are 0 to 255. The label# is a symbolic label corresponding to the address of an instruction. The instruction also can include an optional_token. In this example the optional token can be a defer one value which will execute one instruction following this branch instruction before performing the branch operation. The optional token can alternatively be a defer two instructions which is allowed with the br!=byte instruction and executes the two instructions following this branch instruction before performing the branch operation. The optional token can alternatively be a defer three instructions which is allowed with the br!=byte instruction and which causes the processor to execute the three instructions following this branch instruction before performing the branch operation.

Example: br!=byte[reg, byte_spec, byte_compare_value, label#], defer[3]

This instruction represents an instruction that compares an aligned byte of a register operand to an immediate specified byte value. The byte_spec parameter

represents the aligned byte to compare (0 is right-most byte, 3 is leftmost byte). The ALU condition codes are set by subtracting the specified byte value from the specified register byte. If the values match, the specified branch is taken. There is a 3 cycle branch latency associated with this microword, since condition codes are evaluated and therefore, the branch deferments of 0, 1, 2 or 3 are allowed in order to fill the latency with useful work. The register can be an A or B bank register.

Referring to FIG. 6, the two register address spaces that exist are Locally accessibly registers, and Globally accessible registers accessible by all microengines. The General Purpose Registers (GPRs) are implemented as two separate banks (A bank and B bank) whose addresses are interleaved on a word-by-word basis such that A bank registers have lsb=0, and B bank registers have lsb=1. Each bank is capable of performing a simultaneous read and write to two different words within its bank.

Across banks A and B, the register set 76b is also organized into four windows $76b_0$-$76b_3$ of 32 registers that are relatively addressable per thread. Thus, thread_0 will find its register 0 at 77a (register 0), the thread_1 will find its register_0 at 77b (register 32), thread_2 will find its register_0 at 77c (register 64), and thread_3 at 77d (register 96). Relative addressing is supported so that multiple threads can use the exact same control store and locations but access different windows of register and perform different functions. The uses of register window addressing and bank addressing provide the requisite read bandwidth using only dual ported RAMS in the microengine 22f.

These windowed registers do not have to save data from context switch to context switch so that the normal push and pop of a context swap file or stack is eliminated. Context switching here has a 0 cycle overhead for changing from one context to another. Relative register addressing divides the register banks into windows across the address width of the general purpose register set. Relative addressing allows access any of the windows relative to the starting point of the window. Absolute addressing is also supported in this architecture where any one of the absolute registers may be accessed by any of the threads by providing the exact address of the register.

Addressing of general purpose registers 78 can occur in 2 modes depending on the microword format. The two modes are absolute and relative. In absolute mode, addressing of a register address is directly specified in 7-bit source field (a6-a0 or b6-b0):

```
              7   6   5   4   3   2   1   0
            +---+---+---+---+---+---+---+---+
A GPR:      | a6| 0 | a5| a4| a3| a2| a1| a0|   a6=0
B GPR:      | b6| 1 | b5| b4| b3| b2| b1| b0|   b6=0
SRAM/ASB:| a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=0        SDRAM:
| a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=1
```

register address directly specified in 8-bit dest field (d7-d0):

```
              7   6   5   4   3   2   1   0
            +---+---+---+---+---+---+---+---+
A GPR:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=0, d6=0
B GPR:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=0, d6=1
SRAM/ASB:| d7| d6| d5| d4| d3| d2| d1| d0| d7=1, d6=0, d5=0
SDRAM:      | d7| d6| d5| d4| d3| d2| d1| d0| d7=1, d6=0, d5=1
```

If <a6:a5>=1,1, <b6:b5>=1,1, or <d7:d6>=1,1 then the lower bits are interpreted as a context-relative address field (described below). When a non-relative A or B source address is specified in the A, B absolute field, only the lower half of the SRAM/ASB and SDRAM address spaces can be addressed. Effectively, reading absolute SRAM/SDRAM devices has the effective address space; however, since this restriction does not apply to the dest field, writing the SRAM/SDRAM still uses the full address space.

In relative mode, addresses a specified address is offset within context space as defined by a 5-bit source field (a4-a0 or b4-b0):

```
            7   6   5   4   3   2   1   0
          +---+---+---+---+---+---+---+---+
A GPR:    | a4| 0 |context| a3| a2| a1| a0| a4=0
B GPR:    | b4| 1 |context| b3| b2| b1| b0| b4=0
SRAM/ASB:|ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=0
SDRAM:    |ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=1
```

or as defined by the 6-bit dest field (d5-d0):

```
            7   6   5   4   3   2   1   0
          +---+---+---+---+---+---+---+---+
A GPR:    | d5| d4|context| d3| d2| d1| d0|   d5=0, d4=0
B GPR:    | d5| d4|context| d3| d2| d1| d0|   d5=0, d4=1
SRAM/ASB:| d5| d4| d3|context| d2| d1| d0|   d5=1, d4=0, d3=0
SDRAM:    | d5| d4| d3|context| d2| d1| d0|   d5=1, d4=0, d3=1
```

If <d5:d4>=1,1, then the destination address does not address a valid register, thus, no dest operand is written back.

Other embodiments are within the scope of the appended claims.

What is claimed is:

1. A computer instruction, comprises:

a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte in a register being equal or not equal to a specifed byte value, if the specified byte matches or mismatches the byte value.

2. The instruction of claim 1 wherein the branch is to an instruction at a specified label.

3. The instruction of claim 1 wherein the branch instruction comprises:

a bit_postion field that specifies the byte in a longword contained in the register.

4. The instruction of claim 1 wherein the branch instruction comprises:

an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation.

5. The instruction of claim 1 wherein the branch instruction comprises:

an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation where the number of instructions can be specified as one, two or three.

6. The instruction of claim 1 wherein the register is a context-relative transfer register or a general-purpose register that holds the operand.

7. The instruction of claim 1 wherein the branch instruction comprises:

an optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction.

8. The instruction of claim 1 wherein the branch instruction comprises:

an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation; and

a second optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction.

9. The instruction of claim 1 wherein the branch instruction allows a programmer to which bit of the register to use to determine the branch operation.

10. The instruction of claim 1 wherein the branch instructions allows branches to occur based on evaluation of a byte that is in a data path of a processor.

11. The instruction of claim 1 wherein the branch instruction branches on a byte matching the byte value and wherein the instruction prefetches the instruction for the "branch taken" condition.

12. The instruction of claim 1 wherein the branch instruction branches on a byte not matching the byte value and wherein the instruction prefetches the next sequential instruction.

13. The instruction of claim 1 wherein the branch instruction includes a Byte_spec Number that specifies the byte in the register to be compared with byte_compare_value.

14. A computer program product residing on a computer readable medium for causing a processor that executes multiple contexts to perform a function comprises instructions causing the processor to:

fetch a byte stored in a register;

determine whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and

perform a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register

15. The product of claim 14 wherein the branch is to an instruction at a specified label.

16. The product of claim 14 wherein the program includes a branch instruction that comprises:

a bit_postion field that specifies the byte in a longword contained in the register.

17. A processor comprises:

a register stack;

an arithmetic logic unit coupled to the register stack and a program control store that stores a branch instruction that causes the processor to:

fetch a byte stored in a register;

determine whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and

perform a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register.

18. The processor of claim 17 wherein instructions to perform the branch branch to an instruction at a specified label.

19. The processor of claim 1 wherein a bit_postion field in an instruction specifies the byte in a longword contained in the register.

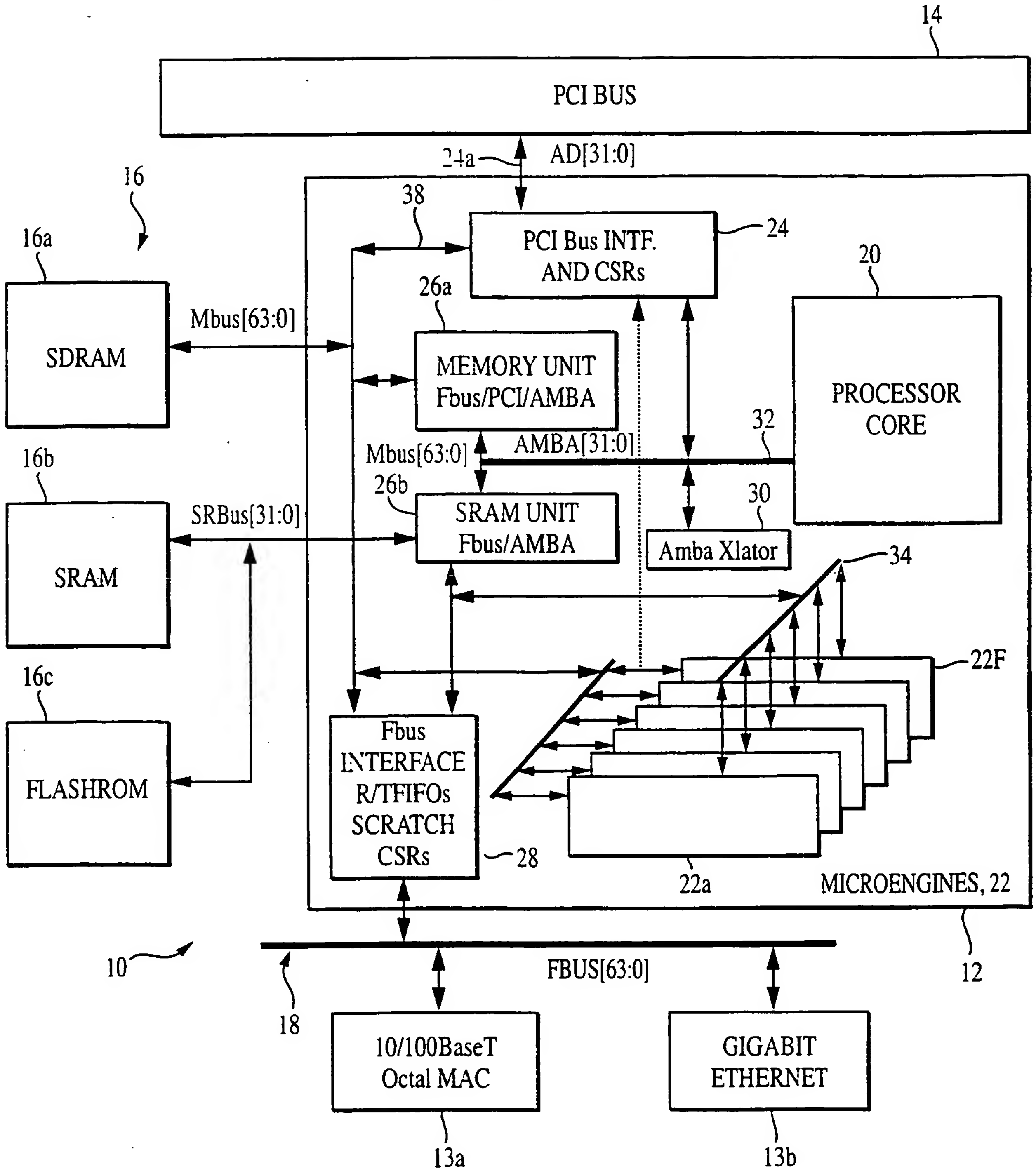
20. A method of operating a processor comprises:

fetching a byte stored in a register;

determining whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and

performing a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register.

21. The method of claim 20 wherein performing the branch branches to an instruction at a specified label.



14
PCI BUS
24a
AD[31:0]
16
38
PCI Bus INTF. AND CSRs
24
16a
20
26a
SDRAM
Mbus[63:0]
MEMORY UNIT Fbus/PCI/AMBA
PROCESSOR CORE
32
Mbus[63:0]
AMBA[31:0]
16b
26b
30
SRAM UNIT Fbus/AMBA
SRBus[31:0]
SRAM
Amba Xlator
34
22F
16c
Fbus INTERFACE R/TFIFOs SCRATCH CSRs
FLASHROM
28
22a
MICROENGINES, 22
10
FBUS[63:0]
12
18
10/100BaseT Octal MAC
GIGABIT ETHERNET
13a
13b

FIG. 1

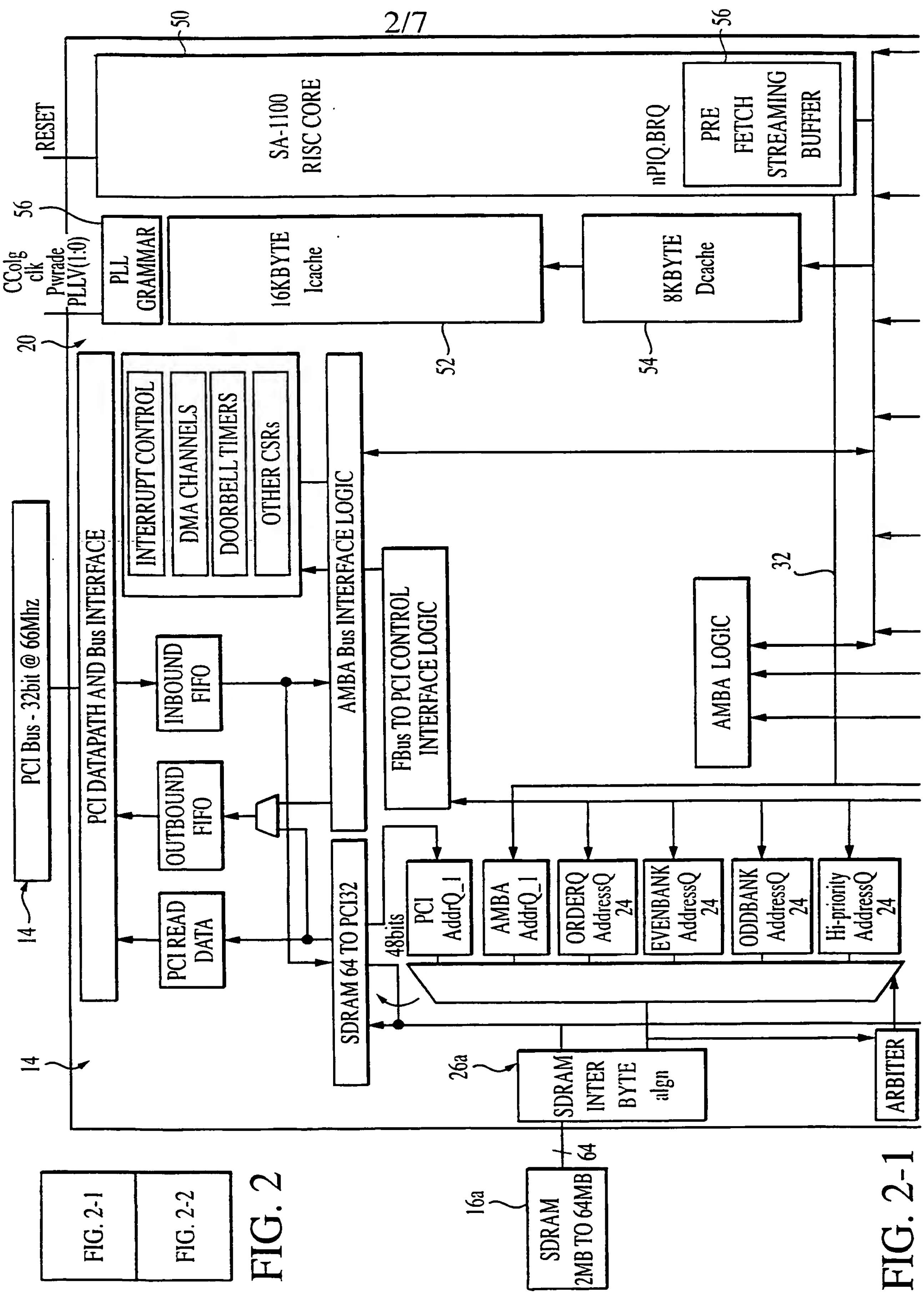

RESET
50
SA-1100
RISC CORE
56
nPIQ.BRQ
PRE
FETCH
STREAMING
BUFFER
CColg
clk
Pwrade
PLLV(1:0)
56
PLL
GRAMMAR
16KBYTE
Icache
52
8KBYTE
Dcache
54
20
PCI Bus - 32bit @ 66Mhz
14
PCI DATAPATH AND Bus INTERFACE
INTERRUPT CONTROL
DMA CHANNELS
DOORBELL TIMERS
OTHER CSRs
AMBA Bus INTERFACE LOGIC
FBus TO PCI CONTROL
INTERFACE LOGIC
32
AMBA LOGIC
INBOUND
FIFO
OUTBOUND
FIFO
PCI READ
DATA
SDRAM 64 TO PCI32
48bits
PCI
AddrQ_1
AMBA
AddrQ_1
ORDERQ
AddressQ
24
EVENBANK
AddressQ
24
ODDBANK
AddressQ
24
Hi-priority
AddressQ
24
14
26a
SDRAM
INTER
BYTE
algn
ARBITER
64
16a
SDRAM
2MB TO 64MB
FIG. 2-1
FIG. 2-2
FIG. 2

FIG. 2-1

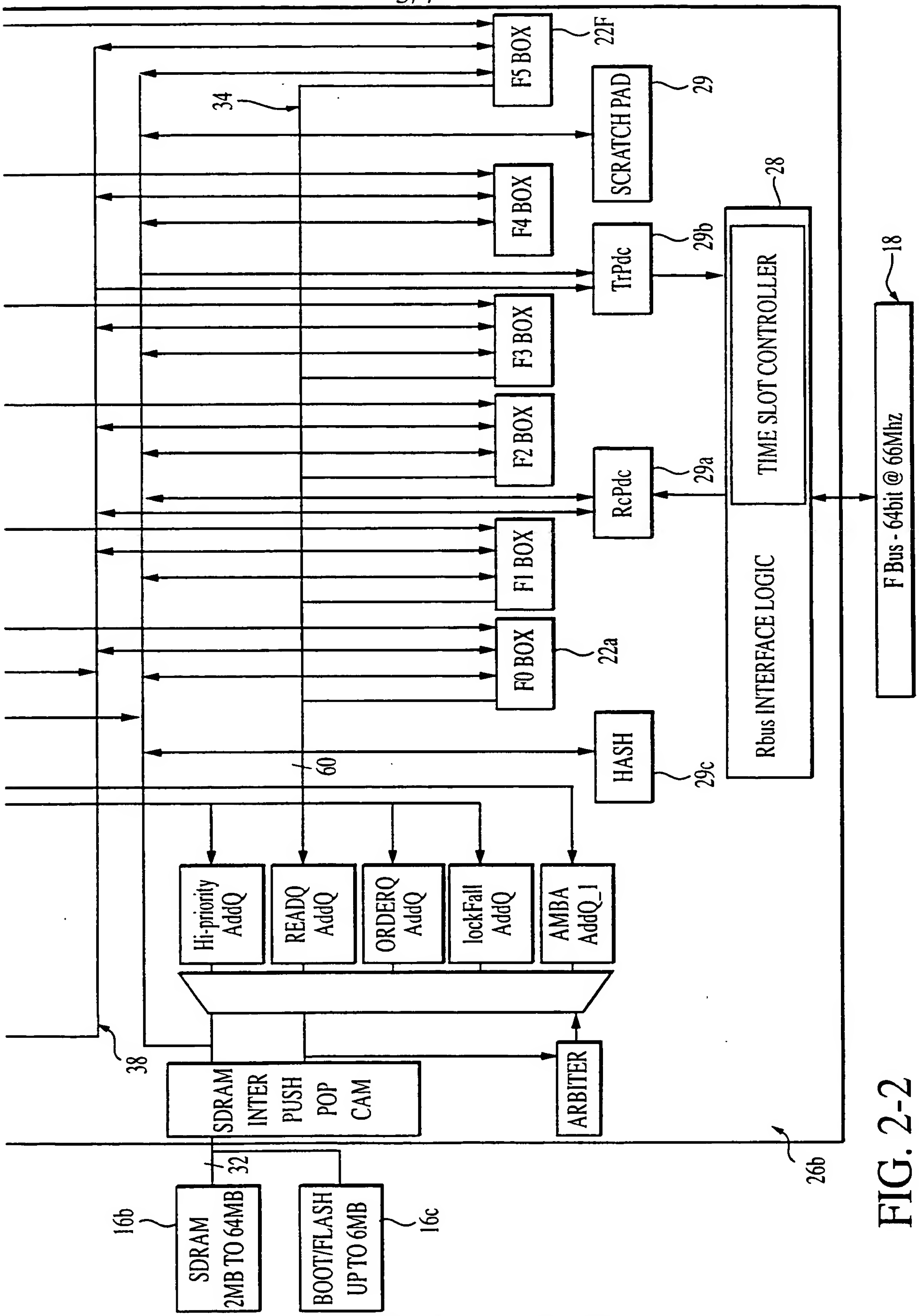

F5 BOX
22F
SCRATCH PAD
29
34
F4 BOX
28
TrPdc
29b
TIME SLOT CONTROLLER
18
F3 BOX
F2 BOX
RcPdc
29a
F Bus - 64bit @ 66Mhz
F1 BOX
Rbus INTERFACE LOGIC
F0 BOX
22a
HASH
29c
60
Hi-priority AddQ
READQ AddQ
ORDERQ AddQ
lockFall AddQ
AMBA AddQ_1
38
SDRAM INTER PUSH POP CAM
ARBITER
26b
32
SDRAM 2MB TO 64MB
16b
BOOT/FLASH UP TO 6MB
16c

FIG. 2-2

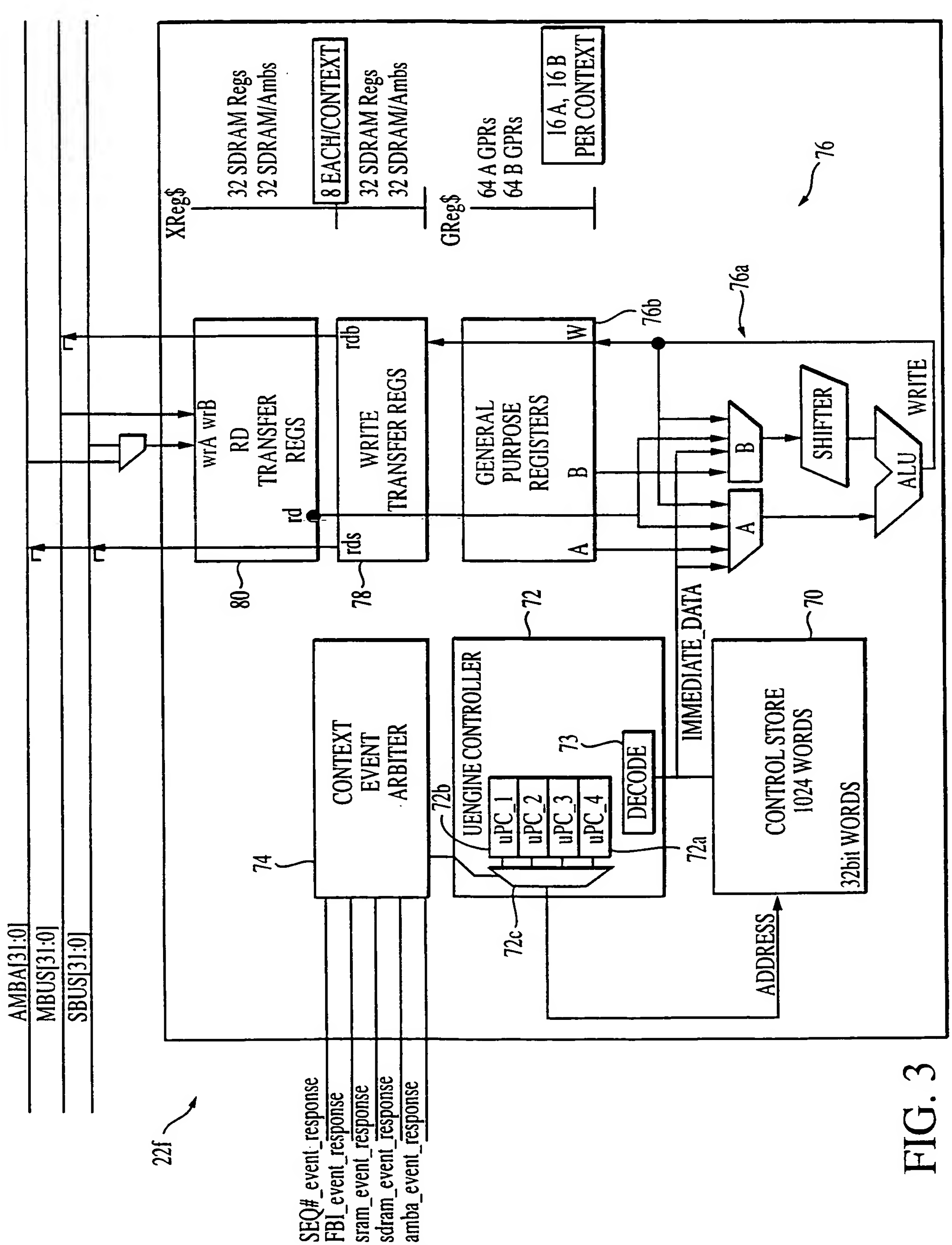

AMBA[31:0]
MBUS[31:0]
SBUS[31:0]
XReg$
32 SDRAM Regs
32 SDRAM/Ambs
8 EACH/CONTEXT
32 SDRAM Regs
32 SDRAM/Ambs
GReg$
64 A GPRs
64 B GPRs
16 A, 16 B PER CONTEXT
76
76a
76b
wrA wrB
RD TRANSFER REGS
rd
rds
rdb
WRITE TRANSFER REGS
GENERAL PURPOSE REGISTERS
W
B
A
SHIFTER
ALU
WRITE
80
78
72
70
IMMEDIATE_DATA
CONTEXT EVENT ARBITER
UENGINE CONTROLLER
uPC_1
uPC_2
uPC_3
uPC_4
DECODE
73
72a
72b
72c
74
CONTROL STORE 1024 WORDS
32bit WORDS
ADDRESS
22f
SEQ#_event_response
FBI_event_response
sram_event_response
sdram_event_response
amba_event_response

FIG. 3

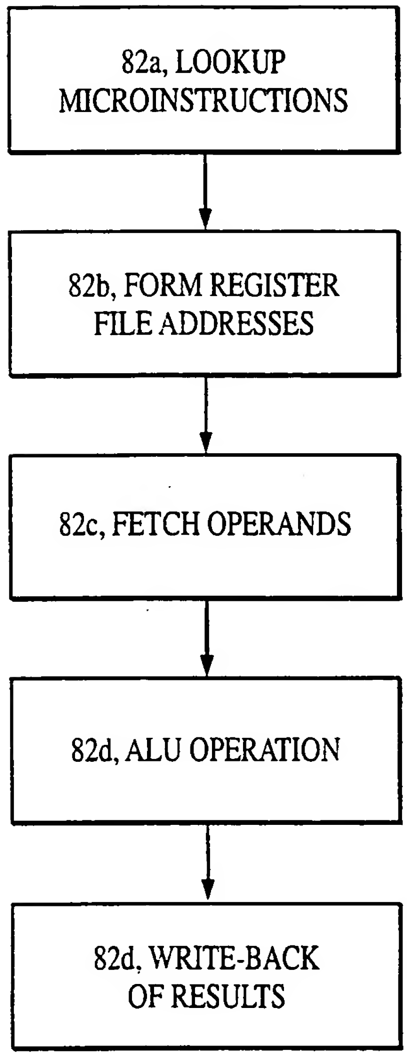

82a, LOOKUP MICROINSTRUCTIONS
82b, FORM REGISTER FILE ADDRESSES
82c, FETCH OPERANDS
82d, ALU OPERATION
82d, WRITE-BACK OF RESULTS

FIG. 4

```
            31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
BR_BYTE     +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
(set cc)    | 0 | 1 | 1 |BShft |A/B Rel Src(aluB) | Hi-Im Data |      Branch Address      | defbr |Eq|Low-Im data|B-A
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
BR_REG      | 0 | 1 | 0 |A/B Abs Src  (for ALUB)| xxxxxxxxxxxx|      Immediate Data      | defbr |xxxxxxxxxxxxxxx|add
            +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Branch Descriptions:

Defbr => gives number of defered instructions (must be less/equal max_allowed (or BR_ev field))

Defered instructions can NOT be branch instructions

Br_Byte => (EQ assumes GT, NEQ assumes NT)

Can have defer = 3 on conditional branchs following br-bit or br-byte

FIG. 5

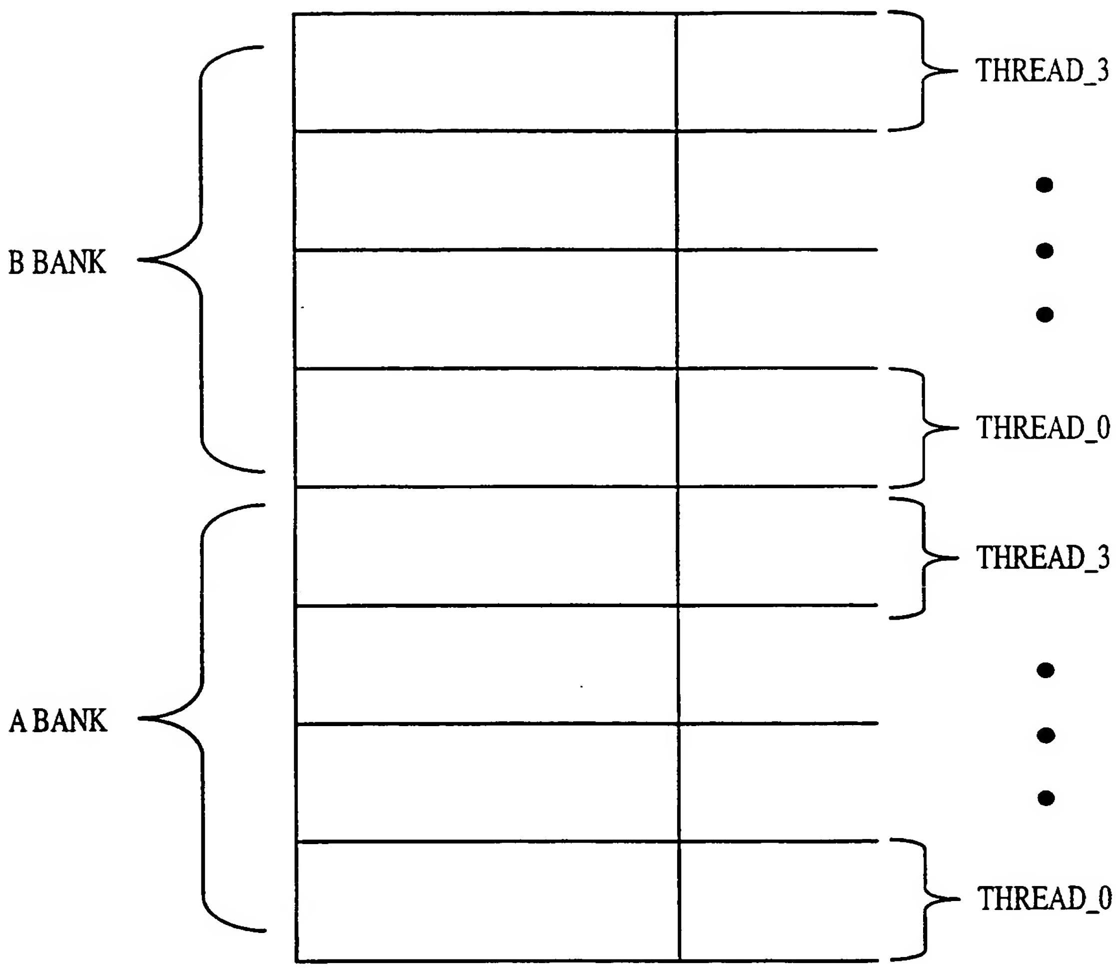


FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/23996

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 9/32, 9/44
US CL : 712/233, 234
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 712/233, 234, 219, 235

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST, IEEE ONLINE
search terms: branch, defered, delayed, programmable, delayed, register, byte

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,073,864 A (METHVIN et al.) 17 December 1991 Abstract	1, 14, 17, 20
A	US 4,392,758 A (BOWLES ET AL.) 12 July 1983 col. 17, Appendix D	1, 14, 17, 20
X	US 4,724,521 A (CARRON et al.) 09 February 1988 col. 117, line 48; col. 134, lines 5-50	1-4, 6, 9, 13-21

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 OCTOBER 2000	15 NOV 2000

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Peggy Hanoel LARRY DONAGHUE
Facsimile No. (703) 305-3230	Telephone No. (703) 305-9675

TENT COOPERATION TREATY

From the
INTERNATIONAL PRELIMINARY EXAMINING AUTHORITY

To: DENIS G. MALONEY
FISH & RICHARSON P.C.
225 FRANKLIN STREET
BOSTON MA 02110-2804

PCT

NOTIFICATION OF TRANSMITTAL OF INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Rule 71.1)

Date of Mailing *(day/month/year)*	05 OCT 2001
Applicant's or agent's file reference 10559-305WO1	IMPORTANT NOTIFICATION

International application No.	International filing date *(day/month/year)*	Priority Date *(day/month/year)*
PCT/US00/23996	31 AUGUST 2000	01 SEPTEMBER 1999

Applicant

INTEL CORPORATION

1. The applicant is hereby notified that this International Preliminary Examining Authority transmits herewith the international preliminary examination report and its annexes, if any, established on the international application.

2. A copy of the report and its annexes, if any, is being transmitted to the International Bureau for communication to all the elected Offices.

3. Where required by any of the elected Offices, the International Bureau will prepare an English translation of the report (but not of any annexes) and will transmit such translation to those Offices.

4. **REMINDER**

The applicant must enter the national phase before each elected Office by performing certain acts (filing translations and paying national fees) within 30 months from the priority date (or later in some Offices)(Article 39(1))(see also the reminder sent by the International Bureau with Form PCT/IB/301).

Where a translation of the international application must be furnished to an elected Office, that translation must contain a translation of any annexes to the international preliminary examination report. It is the applicant's responsibility to prepare and furnish such translation directly to each elected Office concerned.

For further details on the applicable time limits and requirements of the elected Offices, see Volume II of the PCT Applicant's Guide.

Name and mailing address of the IPEA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	LARRY DONAGHUE Peggy Harrod
Facsimile No. (703) 305-3230	Telephone No. (703) 305-9675

Form PCT/IPEA/416 (July 1992)★

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference 10559-305WO1	**FOR FURTHER ACTION**	See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)
International application No. PCT/US00/23996	International filing date *(day/month/year)* 31 AUGUST 2000	Priority date *(day/month/year)* 01 SEPTEMBER 1999
International Patent Classification (IPC) or national classification and IPC IPC(7): G06F 9/32, 9/44 and US Cl.: 712/233, 234		
Applicant INTEL CORPORATION		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 3 sheets.

   ☐ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority. (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of _____ sheets.

3. This report contains indications relating to the following items:

   - I ☒ Basis of the report
   - II ☐ Priority
   - III ☐ Non-establishment of report with regard to novelty, inventive step or industrial applicability
   - IV ☐ Lack of unity of invention
   - V ☒ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
   - VI ☐ Certain documents cited
   - VII ☐ Certain defects in the international application
   - VIII ☐ Certain observations on the international application

Date of submission of the demand 08 MARCH 2001	Date of completion of this report 27 AUGUST 2001
Name and mailing address of the IPEA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer Peggy Harrod LARRY DONAGHUE Telephone No. (703) 305-9675

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/US00/23996

I. Basis of the report

1. With regard to the elements of the international application:*

[X] the international application as originally filed

[X] the description:
pages 1-16, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of ______

[X] the claims:
pages 17-20, as originally filed
pages NONE, as amended (together with any statement) under Article 19
pages NONE, filed with the demand
pages NONE, filed with the letter of ______

[X] the drawings:
pages 1-7, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of ______

[X] the sequence listing part of the description:
pages NONE, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of ______

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.
These elements were available or furnished to this Authority in the following language ______ which is:

[ ] the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).

[ ] the language of publication of the international application (under Rule 48.3(b)).

[ ] the language of the translation furnished for the purposes of international preliminary examination (under Rules 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

[ ] contained in the international application in printed form.

[ ] filed together with the international application in computer readable form.

[ ] furnished subsequently to this Authority in written form.

[ ] furnished subsequently to this Authority in computer readable form.

[ ] The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

[ ] The statement that the information recorded in computer readable form is identical to the writen sequence listing has been furnished.

4. [X] The amendments have resulted in the cancellation of:

[X] the description, pages NONE

[X] the claims, Nos. NONE

[X] the drawings, sheets/~~fig~~ NONE

5. [ ] This report has been drawn as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

* *Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17).*

***Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.*

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.
PCT/US00/23996

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. statement

Novelty (N)	Claims	4-5, 7-8, 11-12	YES
	Claims	1-3, 6, 9-10, 13-21	NO
Inventive Step (IS)	Claims	4-5, 7-8, 11-12	YES
	Claims	1-3, 6, 9-10, 13-21	NO
Industrial Applicability (IA)	Claims	1-21	YES
	Claims	NONE	NO

2. citations and explanations (Rule 70.7)

Claims 1-3,6,9,10 and 13-21 novelty under PCT Article 33(2) as being anticipated by CARRON et al.(4,742,521).

Carron et al. taught the invention, including a branch instruction based on a byte in a register being equal or not equal to a specifed byte (col. 134, lines 5-50), the branch is to an instruction at a specified label (col. 117, line 48); a f ield that specifies the bytes position (col. 134, lines 5-50).

Claims 4-5,7-8 and 11-12 meet the criteria set out in PCT Article 33(2)-(4), because the prior art does not teach or fairly suggest the tokens for the number of delayed instructions executed or a token guess_branch prefetch is performed. Further there is no teaching for the branch instruction to perform a prefetch.

-------------------- NEW CITATIONS --------------------

NONE